(19) **FEDERAL REPUBLIC OF GERMANY**
GERMAN PATENT OFFICE
(12) **Offenlegungsschrift**
(10) **DE 42 05 524 A1**
(21) File reference: P 42 05 524.5
(22) Application date: 24.2.92
(43) Disclosure date: 27.8.92
(51) Int. Cl.[5]:
G 05 B 19/05
B 65 B 57/00
B 65 C 9/40

(30) Union priority: 32, 33, 31
14.11.91 EP 91 11 9483.5
(30) Internal priority: 32, 33, 31
22.02.91 DE 41 05 678.7
(71) Applicant:
Siemens AG, 8000 Munich DE

(72) Inventors:
Bock, Günther, Dipl.-Ing., 8450 Amberg, DE; Macht, Helmut, Dipl.-Ing., 8457 Kümmersbruck, DE; Wombacher, Christof, Dipl.-Ing. (FH), 8450 Amberg, DE; Prechtl, Manfred, Dipl.-Ing. (Univ.), 8470 Nabburg, DE; Lengemann, Andre, Dipl.-Ing. (FH), 8459 Edelsfeld, DE

(54) Stored program control

(57) A new stored program control, particularly for packaging and labelling machines, with several inputs (8) and outputs (9) for the connection of process control elements such as sensors or actuators, is proposed, which has at least a

logic module (10) with an internal interconnection via which at least an output is connected to its corresponding input (E0). A programming method for such a logic module (10) is also presented.

Description

The invention relates to a stored program control, particularly for packaging and labelling machines, with several inputs and outputs for connection of process control elements such as sensors or actuators, as well as a method for operating a stored program control and a method for computer-controlled internal electrical connection of a field programmable gate array.

Previously, machine controllers were built using protective technology. Protective circuits in fact work in parallel and are therefore fast, but they are prone to faults, complicated and are very difficult to build or adapt. The use of stored program controls has now become widespread. These work sequentially and are considerably easier to construct and program. But even modern stored program controls are often not fast enough, for example for controlling packaging and labelling machines, because of their sequential way of working. Controllers for these machines are as a rule also these days constructed on the basis of wired logic elements. This does give these controls a higher processing speed, but the wiring of the logic element is very complicated and fault-prone.

The object of the invention is to provide a stored program control, with which it is possible to deal with extremely fast control processes. Furthermore a method should be indicated that allows such a stored program control, containing a logic module with a field programmable gate array, to be rapidly and easily programmed.

The first object is achieved in that the controller has at least a logic module with an internal interconnection, via which at least an output is connected to its corresponding input. In this way, the output signal of this output is constantly adapted to the value of the corresponding input, so that an involved alarm reaction that would otherwise be necessary on this input can be dispensed with.

The logic module is advantageously designed to work in parallel so that several inputs and outputs can be connected together. This means that the output signals from these outputs are not subject to any fluctuations or variations caused by the processing time, but can be reproduced in a stable manner.

If the internal interconnection of the logic module is programmable and particularly also reprogrammable, the control can be easily adapted to changing requirements.

The programming of the logic module is quite simple here, if the logic module has a – preferably static – memory for storing the conditions that define its internal interconnection. The logic module can particularly be a field programmable gate array (FPGA).

If the control is of a modular design, the logic module is advantageously arranged in an input/output module, because then the system bus does not have to use the control for data transfer. In this case it is also an advantage if the logic module can be directly programmed on the module, e.g. via an interface for connection of a data input/output device, whereby the interface works directly or indirectly on the system bus or also on the logic module itself, or via a plug-in user memory, which defines the internal interconnection of the logic module.

The control works in such a way that at least an input signal is read into a logic module and is processed there, so that from the logic module an output signal corresponding to the input signal can be output.

The second object is achieved by the following method steps:

- from a predefined functional overall behaviour, e.g. on the basis of a predefined functional block diagram, particularly a control system flowchart for a stored program control, internal electrical configurations, and thus connections and if necessary also logic

functions, of the gate array are defined, which bring about the predefined functional overall behaviour, and
- the internal electrical configuration determined in this way, thus connections and if necessary logic functions, are impressed on the gate array,
- whereby independently of the predefined functional overall behaviour when defining the internal electrical connections a part of the, in principle, freely definable internal electrical connections has a fixed specification.

If the logic blocks are broken down into groups by the fixed specification part of the internal electrical connections, which at least in part have the same configurations, regular structures are generated, so that the gate array is broken down virtually into smaller units, namely the groups. In this way it is in fact possible to break down the predefined functional overall behaviour into sub-functions which at least in part can be performed in any group of logic blocks.

As a result, for standard sub-functions, particularly complex standard sub-functions, internal electrical standard configurations, and thus connections and if necessary also logic functions, can be specified, whereby in the individual case the standard sub-functions can also be implemented by more than one group of logic blocks.

If sub-functions and standard sub-functions in the individual case are very simple, if necessary several of these can be combined, provided that the combination into a group of logic blocks is feasible.

The programming of the gate array is then carried out in that

- the sub-functions and/or the standard sub-functions and/or the combination are assigned to the groups of logic blocks,
- taking into account the specified internal electrical connections the internal electrical connections are defined, the groups of logic blocks are interconnected to each other and to

external connections in such a way that the predefined functional overall behaviour is achieved, and

- the internal electrical connections defined in this way are impressed on the gate array, preferably along with the fixed specification internal electrical connections.

The programming of the gate array is particularly simple for the user, if the functional overall behaviour is specified in a programming language for stored program controls, particularly a graphical programming language.

Further advantages and details are evident from the following description of an embodiment, along with the drawings and in association with the other sub-claims. The drawings show as follows:

Fig. 1 – several modules of a modular design stored program control.

Fig. 2 – the internal structure of an input/output module;

Fig. 3 – the connections between inputs and outputs;

Fig. 4 – the structural design of an input/output module;

Fig. 5 – the internal structure of a field programmable gate array;

Fig. 6 – the design of a logic block;

Fig. 7 – an example of a problem to be solved in the form of a traffic light system;

Fig. 8 – the associated circuit engineering implementation of the traffic light system controller;

Fig. 9 – the advance definition of the internal electrical connections;

Fig. 10 – an example of an internal electrical standard connection;

Fig. 11 – the implementation of the specified overall behaviour in the field programmable gate array; and

Fig. 12 – communication between the logic modules and processor, in schematic form.

According to Fig. 1 a modular design stored program control comprises a power supply 1, a central unit 2, the input/output modules 3, 3' and other peripheral units 4. The modules 2, 3, 3', 4 are moreover connected together via a bus 5. The central unit 2 has at least a processor 6 for running a program and an interface 7 for data exchange with a programming device.

As can be seen further from Fig. 1, the module 3 has a logic module 10, which can, for example, be a field programmable gate array (FPGA). The logic module 10 is connected via the processor 11 with the bus 5 and thus also to the central unit 2. In this way it is possible, from a programming device, via the central unit 2, to program the internal interconnection of the logic module 10 in such a way that the inputs 8 are connected via the logic modules 10 according to logic conditions between the input and output signals derived in advance from the program to be run, to the outputs 9. The user generates for this with the abovementioned programming device two program parts: a time-uncritical part and a time-critical part. Both parts are transferred from the programming device to the processor 6 of the central unit 2. The time-uncritical part is stored in the central unit 2 and, as is usual with stored program controls, is run sequentially. The time-critical part is transferred on by the processor 6 to the logic modules 10, 10' and translated by these into a logic circuit.

The two program parts are completely independent of one another. Using special commands it is possible, however, for the processor 6 and the logic modules 10, 10' to exchange information.

Advantageously, moreover, the logic conditions which define the interconnection of the logic module 10, are transferred to a static memory 12 of the logic module 10 and the interconnection of the logic module 10 is defined on the basis of the content of the memory 12.

The module 31 likewise has a logic module 10 with a static memory 12', but the logic module 10' is programmed via a user memory 13'. If the interconnection of the logic module 10' has to be altered, the user memory 13' must be exchanged or reprogrammed, since the logic conditions, which define the interconnection of the logic module 10' are stored in the user memory 13'.

Fig. 2 shows, in a slightly different illustration, the electrical design of the module 3. As can be seen from Fig. 2, the inputs 8 are connected to the logic module 10 via input filters 14 and the outputs 9 to the logic module 10 via output drivers 5. In this way the logic module 10, in the event of an unintentionally incorrect interconnection or a short-circuit or similar malfunctions, is not damaged. Furthermore, via the input filters 14 debouncing of the input signals is possible. Also, via the filters 14 and the drivers 15 the signal level adaptations can be performed, e.g. from 20 mA to 5 V.

The logic module 10 is connected via the bus 16 and the control lines 17 to the processor 11 and thus also with the processor 6. In this way, it is possible to monitor the correct functioning of the logic module 10 – including during operation. In order to monitor the logic module 10 the values of corresponding inputs 8 and outputs 9 can simultaneously be transferred for processing in the logic module 10 to the processor 11 and further to the processor 6. If necessary intermediate states of the logic module 10, e.g. a marker or a

counter state, can also be reported to the processor 6. New control parameters, such as new time constants, can also be transferred to the logic module 10.

The control lines 17 shown in Fig. 2 are, for example, used to transmit a reset signal, with which the internal markers can be reset, and also for the reporting by the logic module 10 to the processor 6 of its current programming state, thus for example the message "Logic module programming changed". At this point mention is made of the fact that the programming of the logic module 10 can only be altered if the logic module is inactive, i.e. if it is not involved in the control of a process. If the logic module 10 consists of various independently functioning parts, it is also possible for only that part whose programming is being changed to be inactive.

According to Fig. 3, the logic module 10 has an input latch 20 and an output latch 21, which for example are clocked by a 1 MHz clock. The inputs 8 are connected to the inputs of the input latch 20. The outputs 9 are connected to the outputs of the output latch 21. Between the latches 20, 21 true parallel processing of the signals takes place. For this, for example, in the logic circuit 22 an elementary logic operation of inputs E0 and E1 is performed, if necessary also with intermediate results, as indicated by the line 23.

The result from the logic circuit 22 can be further processed or also supplied directly to one of the outputs, in the present case output A0. The logic circuits 22 can, as mentioned above, perform elementary logic operations such as, COMPARE, AND, OR, NOT-AND, NOT-OR. In order to be able to perform other more complicated functions, it is an advantage if the logic module 10 has memory elements 24, from which, for example, counters, timers or edge trigger flags can then be constructed.

Fig. 4 shows a preferred structural design of the input/output module 3. As can be seen from Fig. 4, module 3 is a encapsulated printed circuit board, which is connected to a modular design rack 25. The module 3 has a slot 26 for the user module 13', by way of example shown in Fig. 1, and an interface 27 for connecting a programming device. By

means of the user module 13' and the interface 27 it is possible to program the logic module 10 contained in the module 3 directly, i.e. not via the processor 6.

The module 3 also has two sub-D plug-in contacts 28a, 28b, whereby the contacts 28a are used for connecting sensors and the contacts 28b for connecting actuators.

The central idea of the present invention is to model a conventional, sequential user program for a stored program control, as far as possible on the structure known from protective technology, that is to say to wire the corresponding inputs and outputs via logic elements. For this the logic conditions of a user program generated in a programming language for stored program controls is converted into a net-list and stored in a data field. These data are then loaded into the logic module 10 and result there in a corresponding internal interconnection of the logic module 10. Moreover, it is possible, as shown in Fig. 1, to connect several of these logic modules 10, 10' to each other in series and/or in parallel. The program flow is thus distributed between the central unit 2 and the modules 3, 3'.

By means of the direct wiring together of corresponding inputs and outputs the process map that is necessary for conventional stored program controls can be dispensed with. Furthermore, the stored program control is extremely fast, with the "cycle time" tending towards zero. The alarm reaction behaviour is also reproducible, since adherence to the alarm response time is improved.

In the embodiment described above the logic module was used in a modular design automation device. The use in an automation device working independently is equally possible, however. In the minimal version of this automation device the automation device no longer has a processor, but just the logic module, so that the program to be run is executed by the logic module alone. Programming of the logic module takes place in this case either via an interface to a programming device or via a memory module, which has been programmed by the user.

The logic modules 10, 10' are in the present case field programmable gate arrays (FPGAs). Fig. 5 shows a section of the inner structure of such a gate array. The inner structure has a two-dimensional matrix of, for example, 12x12 logic blocks 31. This matrix is surrounded by a ring of input/output blocks. Both the start and the end of each (horizontal) row is assigned to two input/output blocks. The same applies to the (vertical) columns. The input/output blocks are not shown, for the sake of clarity. Furthermore, each row of logic blocks 31 is assigned two uninterruptible connections 32 and each column three connections 33, two of which can be interrupted once in the middle of the column. This arrangement of logic blocks 31 and input/output blocks is interspersed with a network of 13x13 switching matrices 34, whereby adjacent switching matrices 34 are connected to each other by means of five short connections 35.

The logic blocks 31 have, according to Fig. 6, a combinatorial block 310, which from a maximum of 5 input variables 311 determines two output variables 312. Furthermore, the logic block 31 has two flip-flops 313, 314, the input signal of which results either from one of the output variables 312 of the combinatorial block 310, or from a variable directly input via the input 315. The output signals of the flip-flops 313, 314 can either be fed into the combinatorial block 310 or output as one of the output signals 316 of the logic block 31. The logic block 31 can therefore be programmed for which logic and/or memory function it should execute.

The two output functions of the logic block 31 are in principle independent of each other, but in the present case are selected to always be the same, since each of the two outputs 316 can be directly connected to two of the four nearest neighbours to its logic module. Because the two functions are identical, the output signal of each logic block 31 can also have its four nearest neighbours made available as input signals. The result is a more structured topology.

Furthermore, the outputs 316 can be connected to the short connections 35 surrounding them and to the surrounding long connections 32, 33. At the points of intersection between the long connections 32, 33 themselves, and between the long connections 32, 33 and the short connections 35 further electrical connections can also be programmed.

The switching matrices 34 are likewise programmable. They can implement a large number of the theoretically conceivable switching possibilities, such as horizontal and/or vertical through-connections, contacting of horizontal and vertical cable connections 35 and splitting of one connection into two or three.

The input/output blocks are in each case connected to a connecting pin of the chip and can optionally input or output a signal, whereby the signal can be optionally clocked or not.

The programming of the logic blocks 31, the switching matrices 34 and the input/output blocks is in each case stored locally in these elements which have a small static random access memory (SRAM) for this.

With regard to further details of field programmable logic modules, reference is made to the manufacturers' manuals, such as the manuals for the XC 3000 Logic Cell Array Family from Xilinx.

In order to program such gate arrays ASIC design tools exist, by means of which the gate arrays can have their structure programmed with the current path instructions adapted to the gate array. With these instructions, however, the ASIC designer must observe a number of ASIC-specific constraints. Such constraints include, for example, gate times, the signal level of unused gate inputs, and so on. It is obvious that such programming is close to the hardware and highly complex. It can only be handled by true experts.

Programs exist for the translation of the desired programming into internal switchings of the logic module 10. The running time of these programs, i.e. the translation of the desired

overall behaviour into an internal interconnection of the gate array, is, particularly because of the numerous connection possibilities, a few minutes, hours, sometimes even days.

The abovementioned special knowledge is not reasonable for the user of stored program controls, even less so the extremely long running times of the translation programs. The SPC user expects running times of seconds, at the most a few minutes. In the following, therefore, using an example a method is described by which an overall behaviour specified in a programming language in which the SPC user is well versed can be quickly and easily translated into an internal connection of the gate array.

The example is taken from the Siemens AG Simatic S5 Collected Examples, order number E 80850-C 345-X-A1 and is explained using Fig. 7.

"As a result of construction work the traffic along a road has to use a single lane. Since the volume of traffic is very high, temporary traffic lights are installed. When the system is switched on, both sets of lights show red. When an initiator is pressed, the corresponding set of lights turns green after 10 seconds. The green phase should last at least 20 seconds, before through the possible operation of the other initiator both lights show red again. After 10 seconds the other lane is then on green. If no message is present at an initiator, then the light remains in its respective state. Switching off of the system should only be possible after a green phase for a lane. When the control is switched on the initial state (M0) must be set unconditionally".

In order to translate the problem into an SPC programming language initially a renaming of the symbols is carried out as indicated in the following table.

Symbol	Operand	Comment

S0	E0	Switch On (NO contact)
I1	E1	Initiator 1 (NO contact)
I2	E2	Initiator 2 (NO contact)
H1	A1	Green
H2	A2	Green
H3	A3	Red
H4	A4	Red
M0	M0	Initial state M0
M1	M1	State 1
M2	M2	State 2
M3	M3	State 3
M4	M4	State 4
M5	M5	State 5
M6	M6	State 6
M7	M7	State 7
	T1	Time 10 seconds
	T2	Time 20 seconds
	KT 100.1	Time for counter 1
	KT 200.1	Time for counter 2

The associated switching is given in the SPC programming language FUP (Function plan) as shown in Fig. 8. This type of programming is known to the SPC user and frequently used by him. The object is to translate the predefined overall behaviour formulated in an SPC programming language quickly and simply into an FPGA structure, so that the SPC user is ultimately put in a position where he can program the logic module 10 himself.

This is achieved in that the program that translates the SPC user program into the associated internal interconnection of the logic module 10, from the outset uses only a fraction of the theoretically possible complexity of the logic module 10. The way this happens is that part of the in principle freely selectable connections, e.g. the internal

interconnection of the switching matrices 34, has a fixed specification in the translation program, and can therefore not be influenced by the generator of the SPC user program. Specifically, the interconnections of the switching matrices 34 will specify each of the thirteen vertical columns in such a way that on the one hand the top, bottom and middle three switching matrices 34 of a column interconnect to the short connections 35 running horizontally 1:1 and the other short connections 35 for the time being do not connect, and on the other hand the other switching matrices 34 interconnect only the vertical short connections 35 1:1 and block the horizontal short connections 35.

This results in a structure, as shown in Fig. 9: groups 36 are formed, which each contain five logic blocks 31 arranged one below the other and which front and back are each surrounded by five short connections 37 extending over the length of a "half-column". On these groups 36 the circuit from Fig. 8 to be created is modelled in a way and manner that will be explained further on. The two horizontal middle rows of logic blocks 31 are used, again in a way and manner to be explained further on, to generate clock signals.

The resultant groups 36 have a handy size: on the one hand their complexity is low enough and therefore clear enough to be able, in a relatively simple way and manner, to estimate if a sub-network of the overall switching to be created can be implemented by means of one of the groups 36, and on the other the groups are, however, large enough that the overall switching of Fig. 8 does not have to be cut down into sub-networks that are too small. A criterion for selection of the sub-networks, is the connection resources available and the logic capacity of the groups 36. Each sub-network is dimensioned so that it meets the following criteria:

a) it has a maximum of five input signals;
b) it has a maximum of five output signals;
c) a maximum of five logic blocks 31 are required to create the sub-network and
d) the sub-network can be wired within the group 36.

Starting with the OR-gate 81 in Fig. 8 it is immediately apparent that the AND-gate 82 can also be implemented in the same logic block 31, since the combining of these two functions only results in one combinatorial function with three inputs and one output. The RS flip-flop 83 on the other hand is assigned its own logic block 31, since eachof the logic blocks 31, because of a(n arbitrary) compiler specification should only either execute a combinatorial function or perform a memory function. The sub-network 84 can as a result be implemented in one group 36 since in total only four input signals, one output signal and two logic blocks 31 are needed, and the capacity of a group is therefore not exceeded.

Based on similar considerations it is easy to see also that the sub-networks 85 to 88 can each be implemented in one group. The sub-network 90 must, however, be separated from the next network 89, since otherwise the number of inputs exceeds the maximum permitted value of five.

Similarly the other networks 91 to 100 are split among the overall circuit, but are not yet assigned specific groups.

A certain difficulty arises in splitting the individual networks from the creation of the timing elements 99 and 100, since a timing element in the "SPC world" does not have a corresponding counterpart in the "FPGA world".

In order, despite this, to allow the SPC user to program timing elements easily, this function, which is frequently needed for stored program controls (SPCs) is made available to the user as a function macro.

From the compiler running time the compiler recognises that a function macro is present and translates this macro into an internal, standard connection that is movable within the gate array. The internal standard connection would have been described in advance here by the compiler manufacturer or by the ASIC designer. As a resultthe compiler is not to any

significant extent burdened with determining the connections, which implement the function macro.

Fig. 10 shows an example of such a standard connection for a timing element, which can count up to 210 clock cycles. The actual time that can be counted is of course further dependent on the clocking of the counter.

The example shown in Fig. 10 requires three groups 36 of logic blocks 31 alongside each other. Here, the precise modelling of the logic shown in Fig. 10 on FPGA structures is irrelevant for the SPC user. When generating such hard macros, which is performed using standard ASIC design tools, the compiler manufacturer or the ASIC designer must, however, ensure that only local connections, and thus direct connections and short connections 35, are used, but not global long connections 32, 33. In this way, these macros are not only easy to move within the gate array, and thus relocatable, they can also be positioned independently of the networks or macros surrounding them.

Since hard macros are made available to the SPC programmer (or user) by means of a library, and the macros have therefore been created in advance, the internal configuration of such a macro is therefore also not bound by the limited user programming possibilities, but the full complexity of the gate array called upon can be used. User programming restrictions can be dispensed with.

The creation of such hard macros by the compiler manufacturer or the ASIC designer and also the running of the translation program can in fact take hours or even days. In this case, however, this is both possible and tolerable. For to begin with only 3x5 = 15 logic blocks 31 instead of 12x12 = 144 logic blocks 31 have to be connected to each other. And on the other hand the macros, as already mentioned, have been created in advance. As a result the user does not have to create these macros, but they are immediately available to him. The assignment of the hard macros to a certain point in the FPGA, however, lasts for only

fractions of seconds. When designing the hard macro all that has to be ensured is that the four inputs or outputs "start", "reset", "clock" and "timing" are easily accessible.

For other possible standard functions of the "SPC world" larger or smaller such hard macros are of course possible if necessary.

Following a breakdown of the overall circuit into sub-networks 84 to 98 these are combined, provided that the combination also meets the abovementioned criteria a) to d). It may be, for example, that the sub-networks 87 and 94 and sub-networks 93 and 97 can be combined. The step just described is not absolutely necessary, but it does increase the utilization of the gate array.

If unexpectedly in an individual case, in order to create the desired interconnection the number of five inputs or five outputs has to be exceeded, this can be implemented by –as required – leaving free one or more groups 36 before the group 36, requiring more than five input signals, and by way of exception feeding these signals through the horizontal short connections and/or the direct connections of logic blocks 31 of the frontmost group 36 to the group 36 that requires more than five inputs. If these additional connection possibilities are insufficient, an error message is generated "desired circuit cannot be generated, insufficient connection possibilities".

The individual sub-networks 84 to 100 are now assigned to the individual groups 36 as shown in Fig. 11. At this point it is mentioned that the assignment of the sub-networks 84 to 100 to the individual groups 36 was performed in sequence. This is the simplest way and manner in which to carry out any assignment; but more complicated solutions are also conceivable, which already take account of the connections between the sub-networks 84 to 100.

The external groups 36 are not occupied, since the external extended cable connections 37 are not available for connecting these groups, but are needed elsewhere. This use elsewhere will be explained further on.

An example of a more complex solution for the assignment of the sub-networks 84 to 100 to the individual groups 36 would, for example be to arrange the network 98 in the outer right group 36. For this network has as its sole output the process output A4. This process output, however, could be applied directly to an input/output block. Thus neither extended cable connections 37 nor other global connection resources would be needed.

Following the assignment of the sub-networks 84 to 100 to the groups 36 the internal electrical connections are defined. Here, initially, and as far as possible, the direct connections between the logic blocks 31 are used. In the present example of the traffic light circuit there are very few of these; usually only the marker outputs of the sub-networks 84 to 91 are interconnected to the respective adjacent network. Even this is not advisable in the present case, however, as the output signals of the individual sub-networks are also needed elsewhere and therefore in any case more global connections must be resorted to.

Firstly, the input and output signals from and to the process to be controlled are connected, thus the input signals E0 to E2 and the output signals A1 to A4. As far as possible, the input and output signals are fed directly via the horizontal long connections 32 of the most external of the extended cable connections 37. If the horizontal long connections 32 are already occupied, for example because three signals must be connected, but only two horizontal long connections 32 are available, the signals are initially applied to vertical long connections 33 or vertical short connections 37. Then they are fed via a long connection 32 assigned to another row of logic blocks 31 at the edge of the gate array. At the edges of the gate array the input and output signals are further connected by means of the extended short connections 37 in such a way that, for example the logic input signal E0, is connected to the physical process input E0.

By simply counting the remaining internal inputs or outputs of the individual networks 84 to 100, the result is then that, without exception, the five extended short connections 37 between the sub-networks 84 to 100 will always be sufficient to link the inputs and outputs of the sub-networks 84 to 100 vertically with one another. If in an individual case more than five lines are required, for complete connection the vertical long connections 33 would be resorted to, preferably initially the interruptible long connections 33.

Simple counting like this also means that now only thirteen more different signals have to be fed within the gate array, namely the eight marker signals M0 to M7, the two timer signals T1 and T2, and 3 internal signals from sub-network 90 to sub-network 89, from sub-network 93 to sub-network 99 and from sub-network 94 to sub-network 100.

It is now clear, however, that this internal connection is easily possible. For the internal outputs signals are simply applied one after another in the sequence in which there occur to the outer two of the three middle short connection strips 43. In this way, ten internal signals within the entire gate array can be picked off. They are thus available everywhere as internal input signals.

The three internal output signals remaining to be connected are applied to three of the horizontal long connections 32, so that they can likewise be picked off where they are needed. If one of these three signals occurs in the top row of the groups 36 as an output signal, but is needed in the bottom row of groups 36, this problem is resolved in the following way: the respective internal output signal is applied to one of the horizontal long connections 32 in the top half of the gate array, this horizontal connection 32 is connected to a vertical long connection 33 and the vertical long connection 33 is connected to a horizontal long connection 32, which is arranged in the bottom half of the gate array. In this way, the signal is also available in the bottom half of the gate array.

A similar method is obviously used if an internal signal is generated in the bottom half of the gate array, but is needed in the top half as an input signal.

Furthermore, in a forward-looking arrangement of the networks 84 to 100 within the gate array three of the internal signals can be directly connected, so that for the ten internal signals that then remain the outer two of the short connection strips 43 will suffice.

The abovementioned three internal signals each occur only once as an output signal, namely in the sub-networks 90, 93 and 94, and are also needed only once as input signals, namely by sub-networks 89, 99 and 100. If therefore the sub-networks 90 and 89, 93 and 99 and 94 and 100 are each assigned directly one behind the other, these signals can be connected together directly via nearest neighbour connections of the logic blocks 31. Connection via the extended short connection 37 located between the respective sub-network pairs is also possible. In both cases, no horizontal connections 32, 41, 42, 43 are needed. These connections are thus available elsewhere.

For the clocking of the timing elements 99, 100, within the gate array system clocks of 1 ms, 10 ms, 100 ms and 1 second are provided. This takes place in the following way and manner: by means of ASIC design tools the compiler producer will create in advance divider stages, which subdivide any external system clock connected into $1/10^{th}$, $1/100^{th}$ and $1/1{,}000^{th}$ of its original frequency. This macro, referred to as divider macro in the following, is created moreover in such a way that it needs only the two middle and thus far unused rows of logic blocks 31 and the direct connections between these logic blocks. This part of the (system) programming of the FPGAs is fixed and does not change. From outside the logic module 10 via one of the input-output buffers, a clock pulse of 1 ms is directly injected into this divider macro.

The four FPGA-internal system clock pulses of 1, 10, 100 and 1,000 ms are, for example, assigned one to each of the four horizontal long connections 32, which are assigned to the two middle rows of logic blocks 31. These four clocks are thus available throughout the gate array and can accordingly be picked off. The system clock that is connected to the timers 99, 100 is indicated to the compiler by the designation of the input variables KT x.y.

As a general rule x designates the number of clock cycles and y is code for the unit of time. 200.1 means therefore, by way of example, that 200 cycles of the clock with code 1, i.e. 100 ms, have to be counted. The result is that the timer 100 therefore measures 200 x 100 ms = 20 seconds.

For the correct operation of the control program, as a rule the logic modules 10, 10' and the central unit 2 must also exchange data during operation. It may, for example, be that the parameterisation of the logic module 10 has to be changed during operation. Furthermore, the central unit 2 should from time to time at least, be informed of the current state of the logic module 10 (or 10'). The processor 6 and the logic modules 10, 10' are not synchronised with each other, however. A problem therefore arises of data consistency. This problem is further magnified by the fact that data exchange between processor 6 and logic modules 10, 10' takes place serially. Serial data exchange is necessary since otherwise too many pins of the logic modules 10, 10' would be necessary for the data exchange with the processor 6.

The problem is solved by making available to the user further function macros. These function macros implement shift registers, which serve for intermediate storage of input or output data, as well as user memories. Here to begin with the new data to be input are written from the processor 6 into the write buffer store e.g. the logic module 10. During this time the values stored in the buffer stores are in fact available in the logic module 10, but are for the time being not used, since for the present they have not been enabled. By means of an intrinsic command the values newly written to the logic module 10 are transferred from the buffer memory to the user memory. At the same time the values to be read out from the logic module 10 are read into other, so-called read buffer stores. Then the data are read out serially from these read buffer stores into the processor 6.

Fig. 12 shows an example of such a data cycle. In the present case, five lines are needed for transfer of all signals needed. Here the following information is transferred on the lines:

as long as the signal level of the RW line is zero, data can be written to the write buffer stores. As long as the signal level of the RW line is 1, data can be read out from the read buffer stores. All buffer stores are connected to the RW line in such a way that they are triggered on the rising signal edge of the RW line. At the trigger point firstly the data are transferred from the write buffer stores into the user memory. Secondly the data are transferred from the logic blocks 31 to the read buffers stores provided for this purpose.

The signals PA1 and PA2 are address signals. By means of the address signals PA1 and PA2 a maximum of three write buffer stores and read buffer stores each can be addressed. The theoretically possible fourth store (2 signals = $2^2 = 4$ addressing possibilities) should not be used. For these levels are applied to the logic module 10, if no data are being read in or out. Therefore these addresses, e.g. the double zero, should not be used.

CLK is a clock. If CLK is one, the respective responding buffer store reads a new bit in or out.

Data is a data line, on which the information itself is transferred. In the present example (purely random) sheer ones are transferred.

Looked at simply, reading from or writing to the buffer store will require at least four lines, namely the RW, CLK and data lines and at least one address line. These four signals will be applied to the middle, so far unused, of the three short connection strips 43. In this way these four signals are available transversally across the entire logic module 10. If no parameters have to be read in or out anyway, the middle of the three short connection strips 43 is available for the internal connection of the groups 36.

If more than three read buffer stores or write buffer stores have to be addressed, further address signals PA3, PA4, etc., will be applied to the logic module 10. These additional address signals are as a rule applied to two horizontal long connections 32, whereby one of

the long connections is arranged in the top half of the gate array and the other in the bottom half of the gate array.

It goes without saying that the formation of additional memory macros calls for gate array capacities. These gate array capacities are of course no longer available elsewhere.

The abovementioned memory macros have, like the timers, been created in advance by the compiler producer using ASIC design tools. The compiler producer will know from his general technical knowledge how registers and shift registers are constructed. Equally, in the electronics world it will be generally known how to drive shift registers by means of address lines, so that only one is addressed in each case. Such memory configurations therefore require no further explanation in the context of the present invention.

So now all the essential steps for rapid and simple translation of an SPC program into an FPGA structure are known. The internal electrical connections that are now known and the likewise now identified programming of the individual logic blocks 31 are impressed in a known fashion on the gate array, so that it therefore provides the desired overall behaviour, here the traffic light control. Furthermore, the user receives a message concerning the level of use of the gate array or, if the implementation is not possible, a message to this effect and information on why the implementation was notpossible, e.g. because no more spare connections were available.

The traffic light control selected in the present case as an embodiment is of course not as time-critical as other control processes. It was, however, selected since this example allowed a simple explanation of how to proceed.

The result is, therefore, a field programmable gate array that is a long way from being used to the optimum, but whose programming is fast and simple and above all is performed in a way and manner with which the SPC user is familiar.

Claims.

1. Stored program control, particularly for packaging and labelling machines, with several inputs (8) and outputs (9) for the connection of process control elements such as sensors or actuators, characterised in that the control has at least a logic module (10) with an internal interconnection via which at least an output (A0) is connected to its corresponding input (E0).

2. Control according to Claim 1, characterised in that the logic module (10) is designed for parallel operation.

3. Control according to Claim 1 or 2, characterised in that the internal interconnection of the logic module (1) is programmable and, particularly, also reprogrammable.

4. Control according to Claim 1, 2 or 3, characterised in that the logic module (10) has a – preferably static – memory (12) for storing the conditions that define its internal interconnection.

5. Control according to Claim 1, 2, 3, or 4, characterised in that the logic module (10) is a field programmable gate array (FPGA).

6. Control according to Claim 1, 2, 3, 4 or 5, characterised in that the control has at least a processor (6) and a bus (5) connected to the processor (6), whereby the input (8) and output (9) can be connected both to the logic module (10) and to the bus (5) – including simultaneously.

7. Control according to Claim 6, characterised in that the processor (6) is connected via at least a control line (17) to the logic module (10).

8. Control according to one of the above Claims, characterised in that it has at least a user memory (13'), which defines the internal interconnection of the logic module (10').

9. Control according to one of the above claims, characterised in that the logic module (10) has at least a connection (17) for a clock signal.

10. Control according to one of the above claims, characterised in that the input (8) is connected via an input filter (14) and the output (9) via an output driver (15) to the logic module (10).

11. Control according to one of the above claims, characterised in that the control has a modular design and the logic module is arranged in an input/output module (3).

12. Control according to Claim 8 and 11, characterised in that the input/output module (3) has a slot (26) for the user memory (13').

13. Control according to Claim 11 or 12, characterised in that the module (3) has an interface (27) for connection of a data input/output device such as a programming device.

14. Control according to Claim 11, 12 or 13, characterised in that the module has multi-pole contacts (28a, 28b) for connection of the process control elements.

15. Method for operating a stored program control according to one or more of the above claims, characterised in that at least an input signal is read into a logic module (10) and processed there, so that from the logic module (10) an output signal corresponding to the input signal can be output.

16. Method according to Claim 15, characterised in that the reading in, processing and output are clocked.

17. Method according to Claim 15 or 16, characterised in that the input signal is filtered and particularly debounced.

18. Method according to one of Claims 15 to 17, characterised in that the logic module (10) exchanges data with a processor (6) via at least a line (17).

19. Method according to one of Claims 15 to 18, characterised in that for testing of the programming of the logic module (10) at least the input (8) and output (9) are connected to a bus (5).

20. Method according to one of Claims 15 to 19, characterised in that the data exchange between the logic module (10) and the bus (5) takes place via a processor (11) assigned to the logic module (10).

21. Programming method for computer-controlled internal electrical connection of a field programmable gate array, which results from an at least two-dimensional arrangement of logic blocks (31), which can be connected to eachother via internal electrical connections that are freely definable by the user,
- whereby from a predefined functional overall behaviour, e.g. on the basis of a predefined functional block diagram, particularly a control system flowchart for a stored program control, internal electrical configurations, and thus connections and if necessary also logic functions, of the gate array are defined, which bring about the predefined functional overall behaviour,
- whereby the internal electrical configuration determined in this way, thus connections and if necessary logic functions, are impressed on the gate array,

- and whereby independently of the predefined functional overall behaviour when defining the internal electrical connections a part of the, in principle, freely definable internal electrical connections has a fixed specification.

22. Method according to Claim 21, characterised in that the logic blocks (31) are broken down into groups (36) by the fixed specification part of the internal electrical connections, which at least in part have the same configurations.

23. Method according to Claim 22, characterised in that the predefined functional overall behaviour is broken down into sub-functions (84 –100) which at least in part can each be performed in a group (36) of logic blocks (31).

24. Method according to Claim 22 or 23, characterised in that for standard sub-functions (99, 100), particularly complex standard sub-functions (99, 100), internal electrical standard configurations, and thus connections and if necessary also logic functions, can be specified.

25. Method according to Claim 24, characterised in that the standard sub-functions (99,100) can be implemented by more than one group (36) of logic blocks (31).

26. Method according to Claim 23 or 24, characterised in that the sub-functions (84-98) and standard sub-functions (99, 100), which can be implemented in a group (36) of logic blocks (31), are combined, provided that the combination into a group (36) of logic blocks (31) is feasible.

27. Method according to any of Claims 23 to 26, characterised in that
- the sub-functions (84 – 98) and/or the standard sub-functions (99, 100) and/or the combination are assigned the groups (36) of logic blocks (31),

- taking into account the specified internal electrical connections the internal electrical connections are defined, the groups (36) of logic blocks (31) are interconnected to each other in such a way that the predefined functional overall behaviour is achieved, and
- the internal electrical connections defined in this way are impressed on the gate array, preferably along with the fixed specification internal electrical connections.

28. Method according to Claim 27, characterised in that in a gate array with long trajectory long connections (32, 33) and short trajectory short connections (35) the electrical connections with process inputs and process outputs at least in part take place by means of the long connections (32,33).

29. Method according to Claim 27 or 28, characterised in that in a gate array with long trajectory long connections (32, 33) and short trajectory short connections (35) the internal electrical connections as far as possible are made via the short connections (35) and only the internal electrical connections that cannot be made via the short connections are made via the long connections (32, 33).

30. Method according to Claim 29, characterised in that the long connections (32, 33) are at least in part interruptible and in that the internal electrical connections are only then made via the uninterruptible long connections (32, 33), if the internal electrical connections cannot be made via the interruptible long connections (32, 33).

31. Method according to one of Claims 21 to 30, characterised in that the functional overall behaviour is specified in a programming language for stored program controls.

32. Method according to any of Claims 21 to 31, characterised in that the functional overall behaviour is specified in a graphical programming language.

---

14 page(s) of drawings follow
